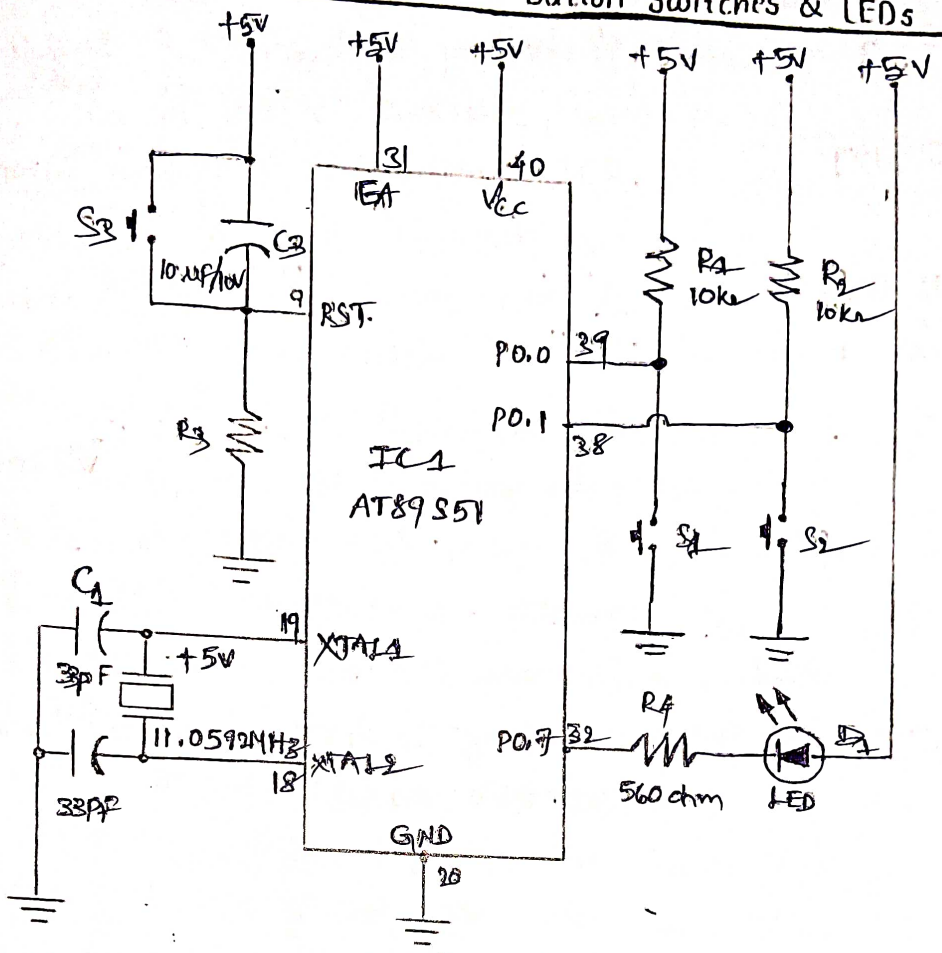
4. INTERFACING SIMPLE I/O DEVICES

4.1 Interfacing Concepts of Push Button Switches & LEDs with 8051

Weightage - 16M



* The circuit diagram for Interfacing push button switch to 8051 is shown above. AT89S51 is the microcontroller used here.
* The circuit is so designed that when push button S1 is depressed the LED D1 goes ON and remains ON until push button switch S2 is depressed, and this cycle is repeated
* Resistor R3, capacitor R3 and push button S3 forms the reset circuitary for the microcontroller.
* Capacitor C1, C2 and crystal X1 belongs to the clock circuitary. R1
* R1 and R2 are pull up resistors for the push buttons
* R4 is the current limiting resistor for LED.

• Program:-

```
        MOV P0, #83H      // Initializing push button switches &
                          initializing LED in OFF state.
READSW: MOV A, P0         // Moving the port value of Accumulator.
        RRC A             // checking the value of Port 0 to know
                          if switch 1 is ON or not.
        JC NXT            // If switch 1 is OFF then jump to NXT
                          to check if switch 2 is ON.
        CLR P0.7          // Turn ON LED because Switch 1 is ON.
        SJMP READSW       // Read switch status again.
NXT : RRC A               // Checking the value of Port 0 to know
                          if switch 2 is ON or not.
        JC READSW         // Jumping to READSW to check status of
                          switch 1 again Provided switch 2 is turned off.
        SETB P0.7         // Turning OFF LED because Switch 2 is ON..
        SJMP READSW       // Jumping to READSW to read status of
                          switch 1 again.
END.
```

• The Logic.:.

* The first instruction - MOV P0, #83H → is to turn LED off [Hex 83 in binary = 10000011] and to initialize switches 1 and 2. switch 1 is connected to port 0.0 and switch 2 is connected to port 0.1. Also note that LED is connected to port 0.7.

Note :. P0.0 = 1 means switch 1 is OFF. and P0.1 = 1 means switch 2 is OFF. P0.0 = 0 means switch 1 is ON and P0.1 = 0 means switch 2 is ON. LED turns ON when P0.7 = 0 & turns OFF when P0.7 = 1. The program has two labels - READSW and NXT.

* It's all about reading switch values - that is P0.0 & P0.1.

* We are using RRC instruction to read switch values. The values of port 0 is moved to accumulator.

* Since port 0 and 1 are used to interface switches 1 and 2, we can get the values of both pord bits in LSB s 0 and 1 of

Scanned with OKEN Scanner

accumulator by using MOV A, P0 instruction.

*RRC - means rotate right through carry.

∴ What RRC do is simple - it will move value of port0.0 to the carry bit. now we can check the carry bit using instruction JC - which means "jump if carry is set". If carry is SET then it means pord 0.0 = 1 and this means switch 1 is OFF. If switch 1 is OFF, then we must check status of the switch 2 and that is why we jump to label NXT.

*In the meantime, if switch 1 is pressed - then value of port0.0 will be equal to zero.

*This will get moved to accumulator and hence an RRC will result in carry bit = 0. If carry bit = 0 then result of executing JC instruction is negative and it will not jump.

*The next instruction will get executed - that is CLR P0.7.

*This clears pord 0.7 to zero and hence LED will turn ON. Once turned On - LED will be kept on until switch 2 is pressed.

*The status of switch 2 is checked in NXT label. When NXT executed, we are using RRC for the second time consecutively.

*This means, the carry bit now holds the value of P0.1 - which is status of switch 2.

*If carry bit = 1 then switch 2 is OFF, means LED should not be turned OFF

*If carry bit = 0 then LED should be turned OFF [SETB P0.7 turns LED OFF]

1.2 Diagram to connect an LED to a Port PIN and an 8051-Assembly Language Program to Blink it with given Time Delay

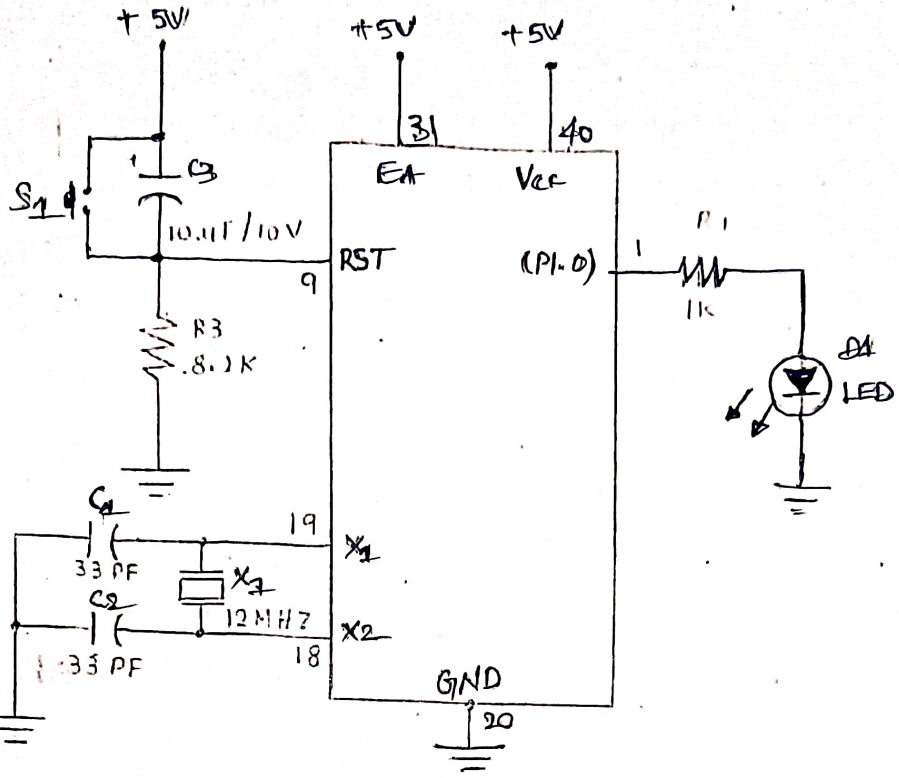
The Microcontroller used here is AT89S51.

*In the circuit, push button switch S1, capacitor C3 & resistor R3 forms the resel circuity.

*When S1 is pressed, voltage at the resel pin (pin9) goes high and this resels the chip.

*C_1, C_2 and X1 are related to the on-chip oscillator which produces the required clock frequency.

* P1.0 (pin1) is selected as the output pin. When P1.0 goes high the transistor Q_1 is forward biased, and LED goes ON.



* When P1.0 goes low the transistor goes to cut off and the LED extinguishes.
* The transistor driver circuit for the LED can be avoided and the LED can be connected directly to the P1.0 pin with a series current limiting resistor (~1k).
* The time for which P1.0 goes high and low (time period of the LED) is determined by the program.

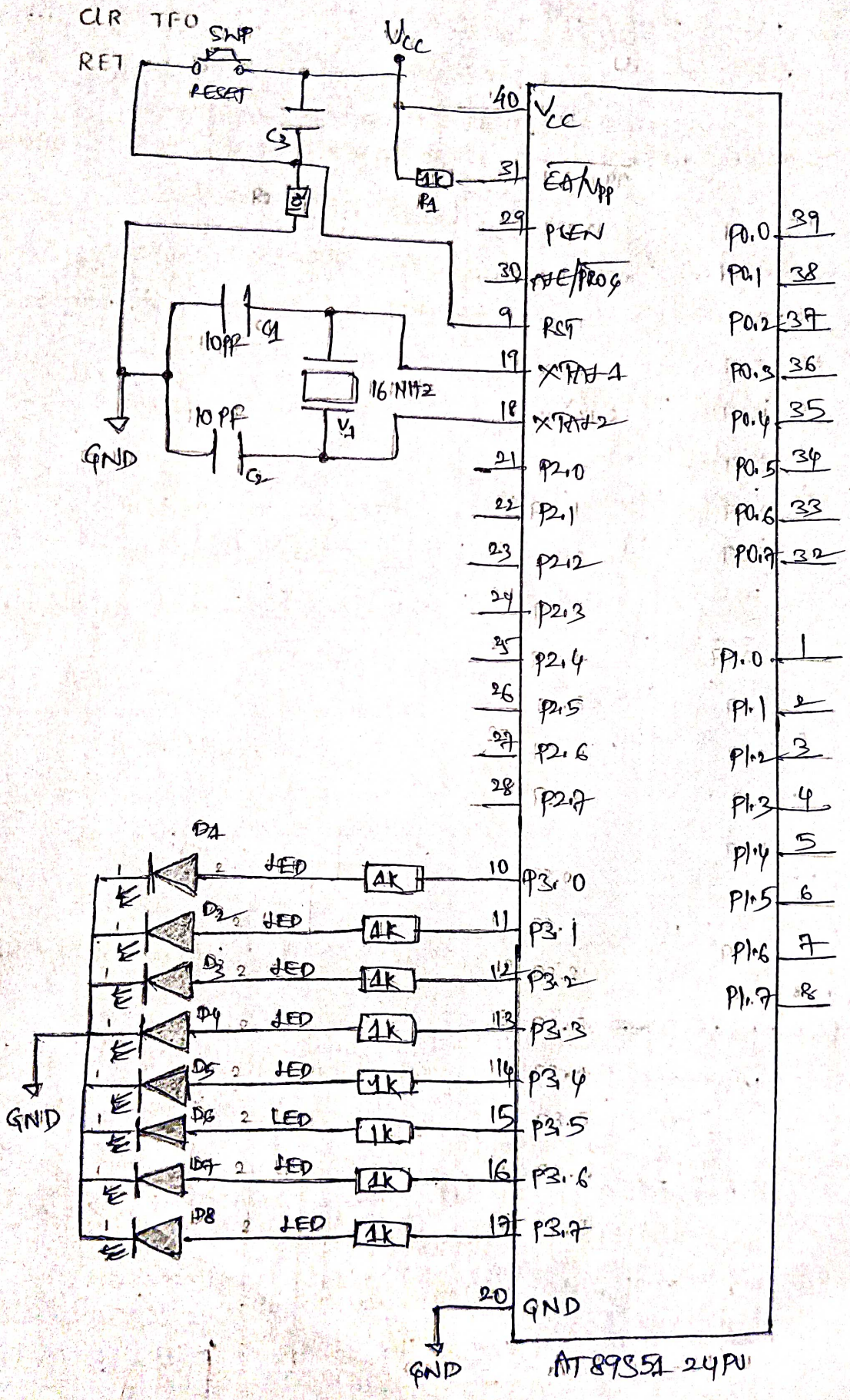
PROGRAM:-

Write an program to blink LED continuosly on a port pin P1.0 with a given time delay of 10m sec.

```
START : CPL  P1.0
        ACALL WAIT
        SJMP  START
WAIT  : MOV  TMOD, #01H
        MOV  TH0, #0DBH
        MOV  TL0, #0FFH
```

Scanned with OKEN Scanner

```
        SETB  TR0
L1      JNB   TF0, L1
        CLR   TR0
        CLR   TF0
        RET
```



PROBLEM - 1

Write an ALP to make all LED's to blink continuosly with a delay of 10 m sec.

```
BACK :MOV A,#0FFH      ; Load Accumulator with FFH (1111 1111)
      MOV P3,A         ; Move content of A into Port3 to
                         turn on all LED's.
      ACALL DELAY      ; Call delay subroutine to generate a
                         delay of 10m sec.
      MOV A,#00H       ; load accumulator with 00H (0000 0000)
      MOV P3,A         ; move the content of A into port 3 to
                         turn off all LEDs.
      ACALL DELAY      ; Call delay subroutine
      SJMP BACK
DELAY : MOV TMOD,#01H  ; Timer 0 in mode 1
      MOV TH0,#0DBH    ; Load TH0 with DBH
      MOV TL0,#0FFH    ; load TL0 with FFH for a delay of 10msec
      SETB TR0
L1    : JNB TF0,L1
      CLR TR0
      CLR TF0
      RET
```

PROBLEM-2

Write an ALP to make all LEDs to blink continuosly with a delay of 20msec

```
BACK : MOV A,#0FFH     ; Load Accumulator with FFH (1111 1111)
      MOV P3,A         ; Move content of A into Port3 to turn on
                         all 8 LED's.
      ACALL DELAY      ; Call delay subroutine to generate a
                         delay of 1sec.
      MOV A,#00H       ; load accumulator with 00H (0000 0000)
      MOV P3,A         ; move the content of A into port 3 to turn
                         of all LED's.
```

```
        ACALL  DELAY  ; Call  delay  subroutine
        SJMP  BACK
DELAY:  MOV   R3, #64H
        MOV  TMOD, #01H; Timer 0 in mode1
        MOV  TH0, #0DBH; Load TH0 with DBH
        MOV  TL0, #00H; load TL0 with FFh for a delay of 10m sec.
        SETB  TR0      ; Start timer 0
L1:     JNB  TF0, L1   ; timer stards counting until overflows,
                         once come out of loop.
        CLR  TR0       ; Clear timer 0 start bit
        CLR  TF0       ; Clear timer 0 overflow bit
        DJNZ  R3, DELAY ; executes DELAY 100 times for a delay
                          of 1sec  RET
```

4.3 Interface a Common Cathode / Anode Seven Segment Display with 8051 and write a program to display a given decimal number

- How does a seven-segment Display work ?

There are two main constructions of the Seven-Segment Display Module. Let's understand both of them.

***Common Anode (CA) 7 segment Display :**

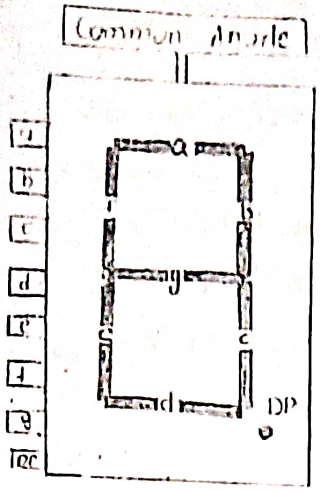
* In the common anode type of 7 Segment Display, the anodes of all the LED's are joined together to VCC supply with a maximum of 10mA current.

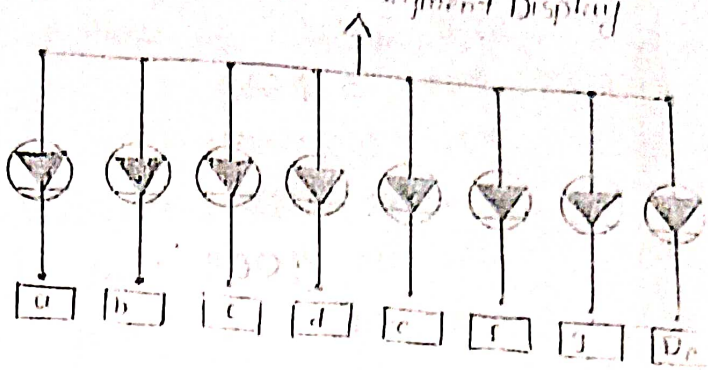
* Whereas the cathodes of these LED's are connected to the I/o ports of the microcontholler.

* As we know that LED turns on when Forward biased and off when reverse biased.

* Now since the anode is connected to +5V to make the LED forward brase, the cathode shoulde be at 0.V means the LED turns ON when a low pluse from the microcontholler is given.

* In summary, to switch on a common anode type's segment, you have to write a "0" to its coressponding pin.





Below table shows the binary / hex values for displaying the digits on Common Anode seven segment display.

Digit	h	g	f	e	d	c	b	a	Hex value
0	1	1	0	0	0	0	0	0	C0
1	1	1	1	1	1	0	0	1	F9
2	1	0	1	0	0	1	0	0	A4
3	1	0	1	1	0	0	0	0	B0
4	1	0	0	1	1	0	0	1	99
5	1	0	0	1	0	0	1	0	92
6	1	0	0	0	0	0	1	0	82
7	1	1	1	1	1	0	0	0	F8
8	1	0	0	0	0	0	0	0	80
9	1	0	0	1	0	0	0	0	90

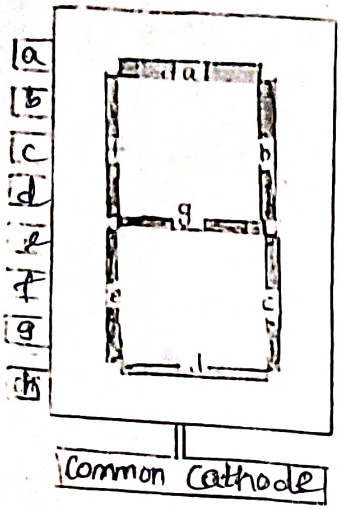
- Common Cathode (CC) 7 Segment Display :-

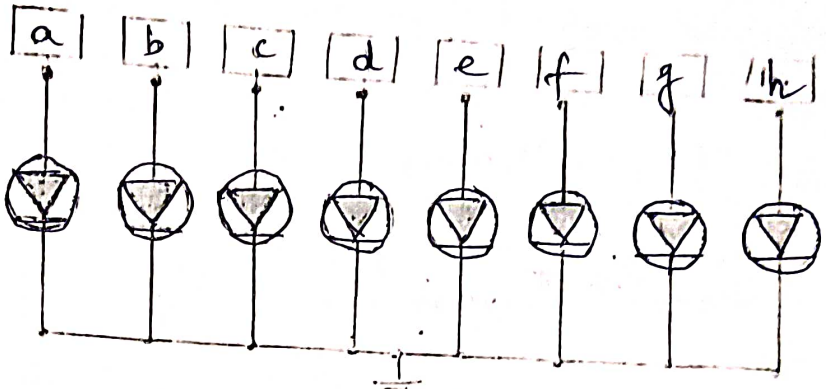
* Even though the internal structure of both Common Anode and Cathode appears the same, as the name suggests, In Common Cathode seven segment display one side, i.e., the cathode part of all the eight light-emitting diodes is shorted together and connected to the ground / GND
* The other side i.e., the anode part, is connected to the microcontroller I/o pins.

*When we give a high pulse through the pins of the microcontroller, the LED turns ON.

*Hence, we can say that a common cathode type display is active High.

*In summary, to switch on a common cathode type's segment, you've to write a "1" to its corresponding pin.

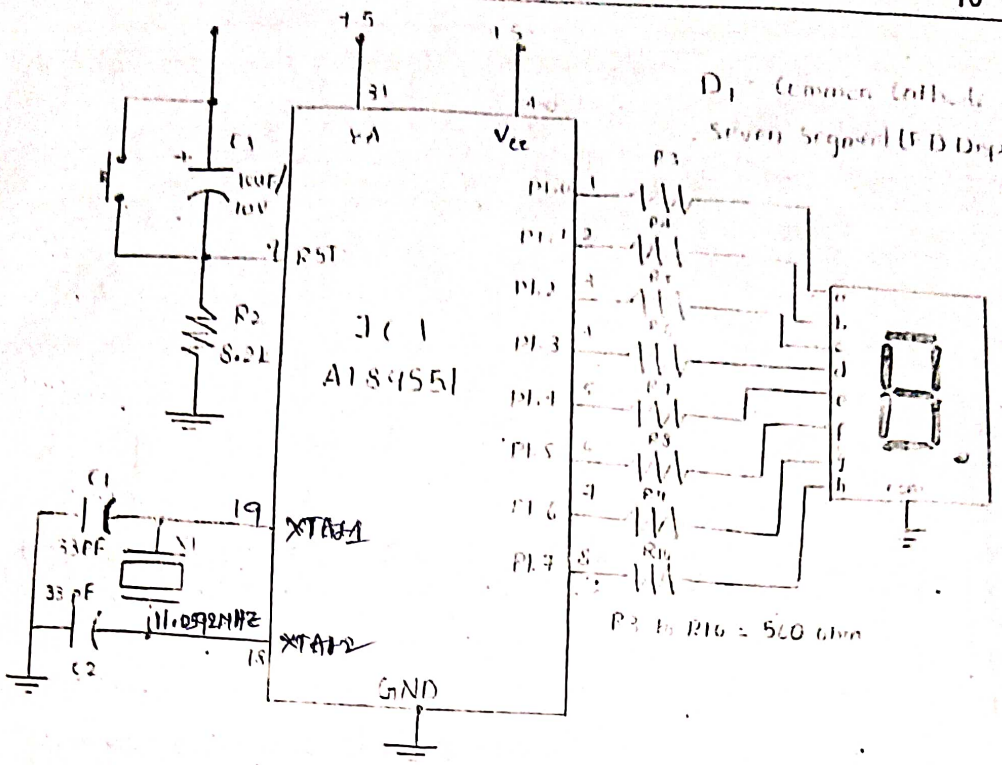




Common cathode 7 segment Display

Below table shows the binary / hex values for displaying the digits on common cathode seven segment display.

Digit	h	g	f	e	d	c	b	a	Hex value
0	0	0	1	1	1	1	1	1	3F
1	0	0	0	0	0	1	1	0	06
2	0	1	0	1	1	0	1	1	5B
3	0	1	0	0	1	1	1	1	4F
4	0	1	1	0	0	1	1	1	66
5	0	1	1	0	1	1	0	1	6D
6	0	1	1	1	1	1	0	1	7D
7	0	0	0	0	0	1	1	1	07
8	0	1	1	1	1	1	1	1	7F
9	0	1	1	0	1	1	1	1	6F



D1 : Common Cathode Seven Segment LED Display

- **Program :**

Write a program to display digit '9' in a common cathode seven segment display.

```
ORG 0000H
        MOV DPTR, #STRING ; point DPTR to first element of string
BACK :  MOV A, #09H       ; initialize accumulator with 09H
        MOVC A, @A+DPTR   ; read data from program memory
                          i.e. dptr + 09H location (Accessed
                          code is 6FH to display 9)
        MOV P1, A         ; send digit value to port 1
        ACALL DELAY       ; call delay subroutine.
HERE :  SJMP HERE
STRING : DB 3FH, 06H, 5BH, 4FH, 66H, 6DH, 7DH, 07H, 7FH, 6FH
DELAY : MOV TMOD, #01H    ; Timer 0 is mode1
        MOV TH0, #0DBH    ; load TH0 with DBH
        MOV TL0, #0FFH    ; load TL0 with FFh for a delay of 10m sec
        SETB TR0
L1 :    JNB TF0, L1
        CLR TR0
        CLR TF0
        RET
```

Scanned with OKEN Scanner

4.4 Functions

4.4. Reasons For The Popularity of LCD's :

1. The declining prices of LCD.
2. The ability to display numbers, characters, & graphics.
3. Incorporation of a refreshing controller into LCD, Thry thereby relieving the CPU of the task of refreshing the LCD.
4. Ease of programming for characters and graphics.

4.5 Functions of Pins of 16X2 LCD Module :

- Pin Discriptions for LCD

Pin	Symbol	I/o	Description
1	Vss	-	Ground
2	Vcc	-	+5v power supply
3	Vee	-	Power supply to control contrast
4	RS	I	RS=0 to select command register, RS=1 to select data regish
5	R/W	I	R/W=0 for write, R/W=1 for read
6	E	I/o	Enable
7	DB0	I/O	The 8 bit data bus.
8	DB1	I/o	The 8 bit data bus
9	DB2	I/o	The 8-bit data bus
10	DB3	I/o	The 8-bit data bus
11	DB4	I/o	The 8-bit data bus
12	DB5	I/o	The 8-bit data bus
13	DB6	I/o.	The 8-bit data bus
14	DB7	I/o	The 8-bit data bus.

- Vcc, Vss, and Vee :- While Vcc and Vss provide 5V and ground, respectively, Vee is used for controlling LCD contrast.
- RS (Register Select :- If RS=0, the instruction command code register is selected, allowing the user to send a command such

as clear display, cursour at home etc. If RS=1 the data register is selected; allowing the user to send data to be displayed on the LCD.

• E (Enable): When data is supplied to data pins, a high-to-low pulse must be applied to this pin in order for the LCD to latch in the data present at the data pins.

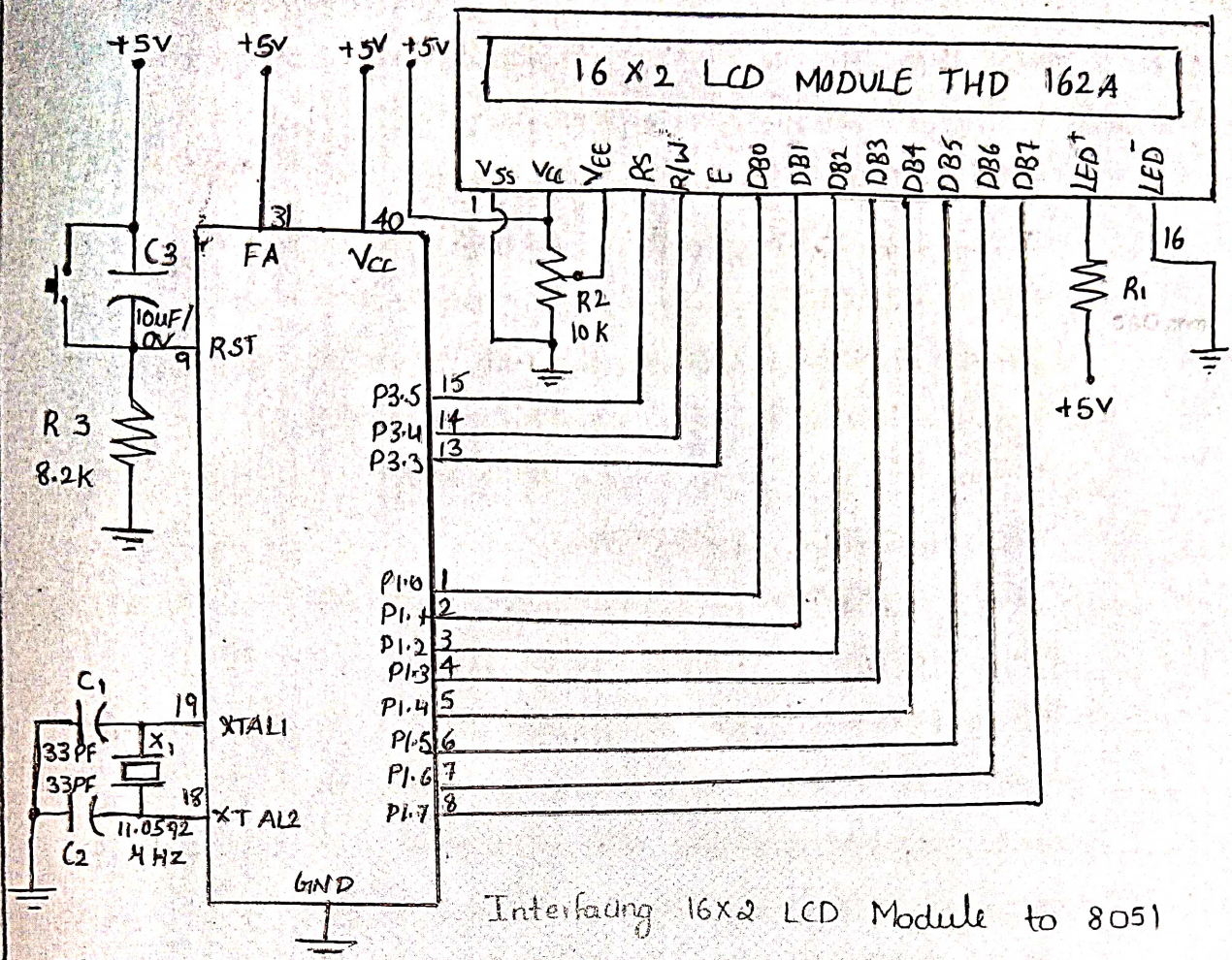
• D0-D7: The 8-bit data pins, D0-D7, are used to send information to the LCD or read the contents of the LCD's internal registers. To display letters and numbers, we send ASCII codes for the letters A-Z, a-z, and numbers 0-9 to these pins while making RS=1.

4.6 Instruction Command codes for Programming 16X2 LCD module.

• LCD Command Codes:

Code (Hex)	Command to LCD Instruction Register
01	Clear display screen
02	Return home
04	Decrement cursor (shift cursor to left)
06	Increment cursor (shift cursor to right)
05	Shift display right.
07	Shift display left.
08	Display off, cursor off
0A	Display off, cursor on
0C	Display on, cursor off
0E	Display on, cursor blinking
0F	LCD on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right.
80	Force cursor to beginning of 1st line
C0	Force cursor to begining of 2nd line
38	2 lines and 5X7 matrix.

4.7 Interfacing of 16X2 LCD MODULE TO 8051



Interfacing 16X2 LCD Module to 8051

* The circuit daigram given above shows how to interface a 16X2 LCD module with AT 89S1 microcontholler. Capacitor C3, resistor R3 and push button switch S1 forms the reset circuitary.
* Ceramic capacitors C1, C2 and crystal X1 is related to the clock circuitry which produces the system clock frequency.
* P1.0 and to P1.7 pins of the microcontholler is connected to the DB0 to DB7 pins of the module respectively and through this route the data goes to the LCD module.
* P3.3, P3.4 and P3.5 are connected to the E, R/W, RS pins of the microcontholler and through this route the control signals are transferred to the LCD module.
* Register R1 limits the current through the back light LED and so do the back light intensity.
* Pot R2 is used for adjusting the contrast of the display.

4.8 8051 ALP DISPLAY GIVEN MESSAGE ON 16X2 LCD MODULE :-

• ALP Display Message on 16X2 LCD Module.

```
ORG 0000H
        MOV A, #38H    ; init. LCD 2 lines, 5X7 matrix
        ACALL COMNWRT  ; call command subroutine.
        ACALL DELAY    ; give LCD some time.
        MOV A, #0EH    ; display, cursor on.
        ACALL COMNWRT  ; call command subroutine
        ACALL DELAY    ; give LCD some time
        MOV A, #01H    ; clear LCD
        ACALL COMNWRT  ; call command subroutine
        ACALL DELAY    ; give LCD some time
        MOV A, #06H    ; shift cursor right
        ACALL COMNWRT  ; call command subroutine
        ACALL DELAY    ; give LCD some time
        MOV A, #84H    ; cursor at line 1, pos. 4
        ACALL COMNWRT  ; call command subroutine
        ACALL DELAY    ; give LCD some time
        MOV A, #'A'    ; display letter A
        ACALL DATAWRT  ; call display subroutine.
        ACALL DELAY    ; give LCD some time.
        MOV A, #'F'    ; display letter F
        ACALL DATAWRT  ; call display subroutine
        ACALL DELAY    ; give LCD some time
        MOV A, #'S'    ; display letter S
        ACALL DATAWRT  ; call display subroutine
        ACALL DELAY    ; give LCD some time
        MOV A, #'H'    ; display letter H
        ACALL DATAWRT  ; call display subroutine
        ACALL DELAY    ; call display subroutine give LCD some time
```

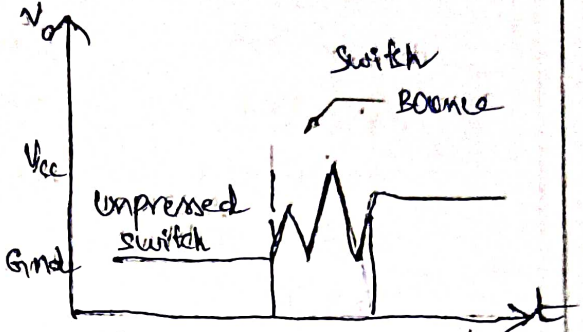
```
        MOV  A, #'A'      ; display letter A
        ACALL DATAWRT     ; call 'delay' subroutine
AGAIN : SJMP  AGAIN       ; Stay here
COMNWRT:                  ; send command to LCD
        MOV P1, A         ; copy reg A to port1
        CLR  P3.5         ; RS=0 for command
        CLR P3.4          ; R/W =0 for write
        SETB P3.3         ; E=1 for high pulse
        ACALL DELAY       ; give LCD some time
        CLR P3.3          ; E=0 for H-to-L pulse
        RET               ; Return from subroutine
DATAWRT:                  ; write data to LCD
        MOV P1, A         ; copy reg A to port1
        SETB P3.5         ; RS=1 for data
        CLR P3.4          ; R/W=0 for write
        SETB P3.3         ; E=1 for high pulse
        ACALL DELAY       ; give LCD some time
        CLR P3.3          ; E=0 for H-to-L pulse
        RET               ; return from subroutine.
DELAY : MOV R3, #50       ; 50 or higher for fast CPU's
HERE 2 : MOV R4, #255     ; R4=255
HERE  : DJNZ R4, HERE     ; Stay until R4 become 0.
        DJNZ R3, HERE2
        RET
        END
```

4.9 KEY BOUNCING PROBLEM AND DE-BOUNCING SOLUTIONS

- Key Bouncing: When we press a pushbutton or toggle switch or a micro switch, two metal parts come into contact to short the supply.

* But they dont connect instantly but the metal parts connect and disconnect several times before the actual stable connection is made.

* The same thing happens while releasing the button.
* This results for the **False Triggering or Multiple Triggering** like the button is pressed multiple times.
* Its like falling a bouncy ball from a height and it keeps bouncing on the surface, until it comes rest.
* Simply, we can say that the **Switch Bouncing** is the non-ideal behaviour of any switch which generates **Multiple Transitions of a Single Input.**



* Switch bouncy is not a major problem what we deal with the power circuits, but it causes problems while we are dealing with the logic or digital circuits.
* Hence, to remove the bounding from the circuit **Switch Debouncing circuit** is used.

Debouncing techniques :-

i). Hardware Debouncing technique Methods :
ii) Software Debouncing.

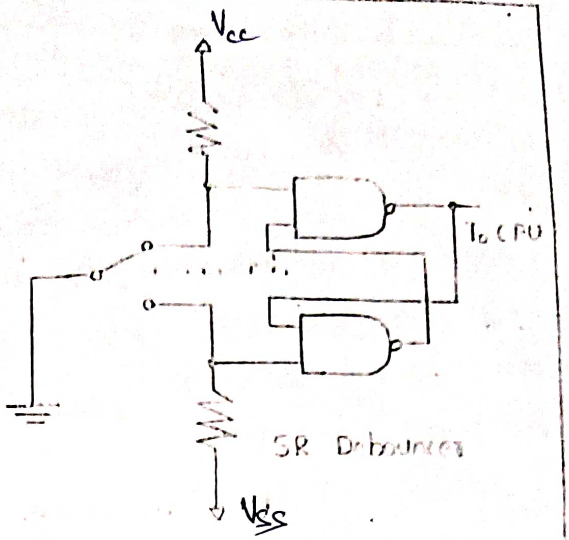
1. Hardware Debouncing Methods :-

a. SR debouncing
b. RC debouncing
c. Switch debouncing IC

a). SR. Debouncing :-

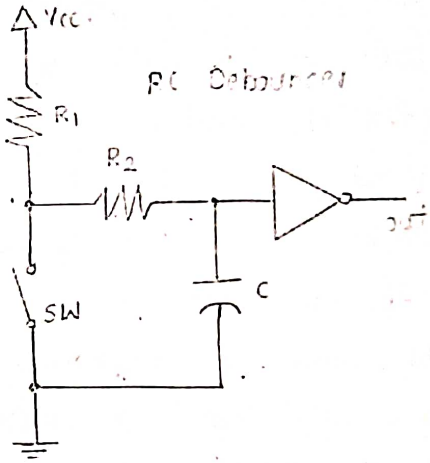
* Hardware debouncing technique uses an S-R latch to avoid bounces in the circuit along with the pull up resistors.
* SR circuit is most effective of all debouncing techniques.
* If the switch is in position a shown in the figure the output of the upper gate is 1 irrespective of the input of the other gate and the one created by the bottom pull up resistor which drives the lower NAND gate to zero which in return races back to the other gate.

* If the switch moves back and forth between the wiads and his is suspended for a while in neither region between the terminals, the latch maintains its state because 0 from the bottom NAND gate is fed back.
* The switch may move between the contacts, but the latchs output ensures it never bangs back and thus switch is bounce free.



b) R-C Debouncing:-

* The S-R circuit is common, but the bulkiness of the circuit causes it to be used rarely also SPDT switches are costlier than SPST (Single Pole Single Throw) Switch.
* Another method of debouncing is to use a RC-circuit.
* The basic R-C circuit used for debugging debouncing is shown here.



* The circuit uses two Resistors, Capacitor, Schmitt trigger hex inverter, SPST switch.

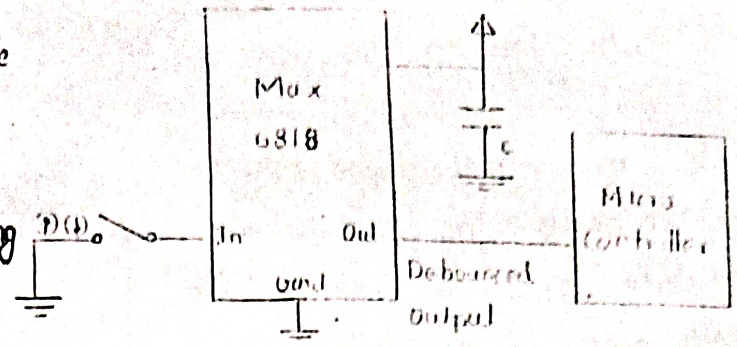
→ *If the switch is open, the voltage across capacitor which is initially zero now charges to Vcc through the R1 & R2.
* The voltage at the input of the Schmitt trigger hex inverter is high hence the output of the inverting Schmitt trigger is low (logic 0).

→ If the switch is closed the capacitor discharges to zero hence the voltage at the input of the Schmitt trigger hex inverter is 0 and output of the inverting Schmitt trigger is high (logic 1).
During the bouncing condition the capacitor will stop the voltage at Vin when it reaches either Vcc or Gnd.

Scanned with OKEN Scanner

C. Switch Debouncing IC:

* These are ICs available in market for switch debouncing.
* Some of the debouncing IC's are MAX6816, MAX 6817, MAX 6818, MC 14490, and LS118.



Above is the circuit diagram for switch debouncing using MAX6818

Software Switch Debouncing:

* In this method, the switches bouncing state effect is eliminated using various algorithms and filters. The programmer can design an algorith with use of shift register and counter such that it will regrster the switch's state after a delay.
* Another method is to use filter algorithms on the sampled input from the switch and determine the state of switch based on the output of such digital filter.
* All this can make the software slighlly inefficent, adding to delay in performance is not implemented correctly.

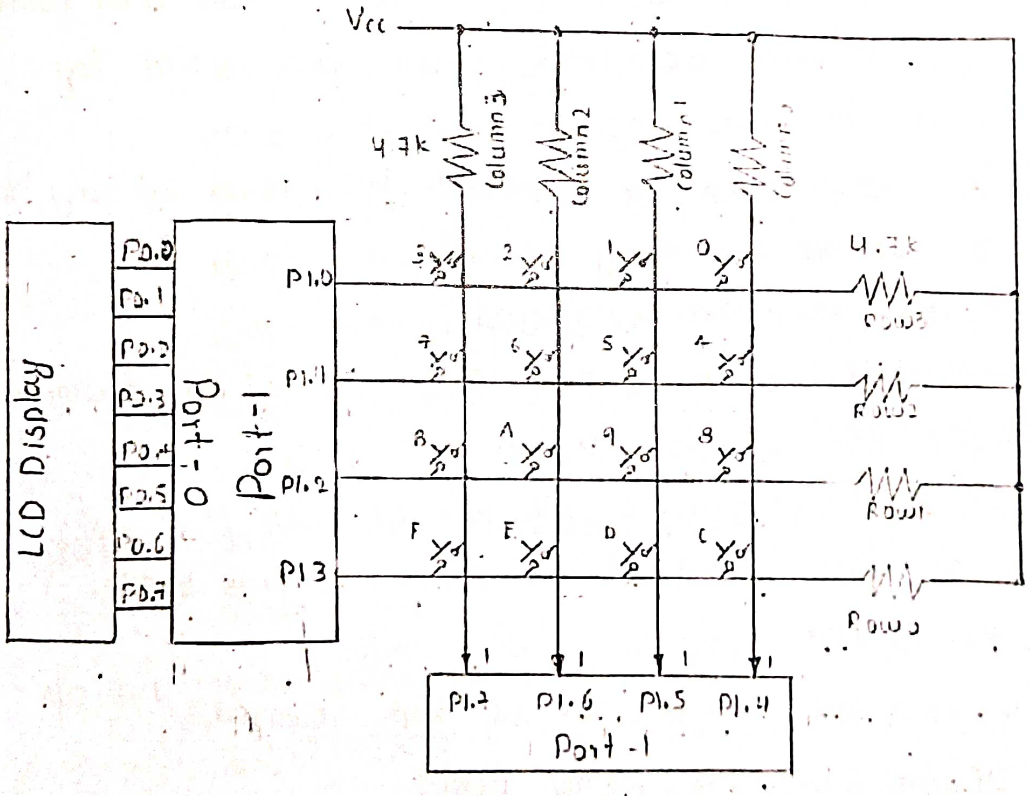
The algorithm for software debouncing is shown.

Counter Method:

1. Initially set up a count value to zero.
2. Using a timer set up a sampling event with a period (Ans 1ms)
3. On the sample event.
4. If switch signal high, then
5. Set count = 0.
6. Set internal switch state released.
7. else
8. Increment count to a max of 10
9. end if
10. If count = 10 then
11. Set internal switch state to pressed.
12. end if

4.10 INTERFACING CONCEPTS OF 4x4 MATRIX KEYBOARD WITH 8051 WITH DIAGRAM

* Below figure shows a 4x4 matrix connected to two ports.
* The rows are connected to an output port and the columns are connected to an input port.
* If no key has been pressed, reading the input port will yield 1s for all columns since they are all connected to high(Vcc).
* If all the rows are grounded and a key is pressed, one of the columns will have 0 since the key pressed provides the path to ground.
* It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed. How it is done is explained below.



- Grounding Rows and Reading the Columns:

* To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch, then reads the columns.
* If the data read from the columns is D3-D0 = 1111, no key has been pressed and the process continuous until a key is pressed.

* However, if one of the column bits has a zero, this means that a key press has occured.
* For example, if D3-D0 = 1101, this means that a key in the D1 column has been pressed.
* After a key press is detected, the microcontroller will go through the process of identifying the key.
* Starting with the top row, the microcontroller grounds it by providing a low to row D0 only; then it reads the columns.
* If the data read is all 1s, no key in that row is activated and the process is moved to the next row.
* It grounds the next row, reads the columns, and checks for any zero.
* This process continues until the row is identified.
* After identification of the row in which the key has been passed, the next task is to find out which column the pressed key belongs to.
* This should be easy since the microcontroller knows at any time which row and column are being accessed.

• Program to Access 4X4 Matrix keyboard :

```
; P1.0 - P1.3 connected to rows, P2.0 - P2.3 connected to columns.
        MOV  P2, #0FFH      ;make P2 an input port.
K1 :    MOV  P1, #00H       ; ground all rows at once.
        MOV  A , P2         ;read all col. Ensure all keys open.
        ANL  A,#00001111B   ; masked unused bits.
        CJNE A,#0000H1111B,K1; check till all keys released.
K2 :    ACALL DELAY         ; Call 20 ms delay.
        MOV  A , P2         ; see if any key is pressed.
        ANL  A, #00001111B  ; mask unused bits
        CJNE A,#00001111B,OVER ; key pressed , await closure
        SJMP K2             ; check if key pressed.
OVER :  ACALL DELAY         ; wait 20ms debounce
```

```
        MOV  A, P2          ; Check key closure
        ANL  A,#00001111B   ; mask unused bits
        CJNE A,#00001111B,OVER1; key pressed, find row
        SJMP K2             ; if none, keep polling
OVER 1: Mov P1,#11111110B   ; ground row-0
        MOV  A, P2          ; read all columns
        ANL  A,#00001111B   ; mask unused bits
        CJNE A,#00001111B,ROW_0; key row 0, find the col.
        Mov P1,#11111101B   ; ground row-1
        MOV  A,P2           ; read all columns
        ANL  A,#00001111B   ; mask unused bits
        CJNE A,#00001111B,ROW_1; key row1, find the col.
        Mov P1, #11111011B  ; ground row-2
        MOV  A, P2          ; read all columns
        ANL  A,#00001111B   ; mask unused bits
        CJNE A,#00001111B,ROW_2; key row2, find the col.
        MOV  P1,#11110111B  ; ground row-3
        MOV  A,P2           ; read all columns
        ANL  A,#00001111B   ; mask unused bits.
        CJNE A,#00001111B,ROW3; key row 3, find the col
        LJMP K2             ; if none, false input, repeat
ROW_0 : MOV DPTR,#KCODE0    ; set DPTR = stat of row0
        SJMP FIND           ; find col key belongs to
ROW_1 : MOV DPTR,#KCODE1    ; set DPTR = start of row 1
        SJMP FIND           ; find col key belongs to
ROW_2 : MOV DPTR,#KCODE2    ; set DPTR = start of row 2
        SJMP FIND           ; find col key belongs to
ROW_3 : MOV DPTR,#KCODE3    ; set DPTR = start of row3
        SJMP FIND           ; find col key belongs to
FIND  : RRC A               ; see if any CY bit is low
        JNC MATCH           ; if zero, get ASCII code
        INC DPTR            ; point to the next column address.
```

```
        SJMP  FIND          ; keep searching
MATCH : CLR A               ; set A=0 (match is found)
        MOVC A, @A+DPTR     ; get ASCII code from table
        MOV P0, A           ; display pressed key.
        LJMP K1
; ASCII LOOK UP TABLE FOR EACH ROW
        ORG 300H
KCODE0: DB '0', '1', '2', '3'    ; ROW 0
KCODE0: DB '4', '5', '6', '7'    ; ROW 1
KCODE0: DB '8', '9', 'A', 'B'    ; ROW 2
KCODE0: DB 'C', 'D', 'E', 'F'    ; ROW 3
        END
```

```
        MOV DPTR, #LUT
        MOV A, #00H
        MOV P0, A
BACK :  MOV A, #0FFH
        MOV P1, A
        CLR P1.0
        JB P1.4, NXT
        MOV A, #00H
        ACALL DISPLAY
NXT1 :  JB P1.5, NXT2
        MOV A, #01H
        ACALL DISPLAY
NXT2 :  JB P1.6, NXT3
        MOV A, #02H
        ACALL DISPLAY
NXT3 :  JB P1.3, NXT4
        MOV A, #03H
        ACALL DELAY
NXT4 :  SETB P1.0
        CLR P1.1
        JB P1.4, NXT5
        MOV A, #04H
        ACALL DELAY
NXT5 :  JB P1.5, NXT6
        MOV A, #05H
        ACALL DISPLAY
NXT6 :  JB P1.6, NXT7
        MOV A, #06H
        ACALL DELAY
NXT7 :  JB P1.7, NXT8
        MOV A, #07H
```

```
        ACALL DISPLAY
NXT8:   SET   P1.1
        CLR   P1.2
        JB    P1.4, NXT9
        MOV   A, #08H
        ACALL DISPLAY
NXT9:   JB    P1.5, NXT10
        MOV   A, #09H
        ACALL DISPLAY
NXT10:  JB    P1.6, NXT11
        MOV   A, #0AH
        ACALL DISPLAY
NXT11:  JB    P1.7, NXT12
        MOV   A, #0BH
        ACALL DISPLAY
NXT12:  SET B P1.2
        CLR   P1.3
        JB    P1.4, NXT13
        MOV   A, #0CH
        ACALL DISPLAY
NXT13:  JB    P1.5, NXT14
        MOV   A, #0DH
        ACALL DELAY
NXT14:  JB    P1.6, NXT15
        MOV   A, #0EH
        ACALL DELAY
NXT15:  JB    P1.7, NXT16
        MOV   A, #0FH
        ACALL DISPLAY
NXT16:  SJMP  BACK
DISPLAY: MOV  A, @A+DPTR
        MOV   P0, A
        RET
```

```
ASCII LOOK UP TABLE FOR EACH RO
        ORG 300H
KCODE0:DB '0', '1', '2', '3' ; ROW3
KCODE0:DB '4', '5', '6', '7' ; ROW2
KCODE0:DB '8', '9', 'A', 'B' ; ROW1
KCODE0:DB 'C', 'D', 'E', 'F' ; ROW0
        END
```